Schedulability conditions for non-preemptive hard real-time tasks with strict period

Mohamed Marouf, Yves Sorel

Schedulability conditions for non-preemptive hard real-time tasks with strict period

Mohamed MAROUF
INRIA Rocquencourt
Domaine de Voluceau BP 105
78153 Le Chesnay Cedex - France
Email: mohamed.marouf@inria.fr

Yves SOREL
INRIA Rocquencourt
Domaine de Voluceau BP 105
78153 Le Chesnay Cedex - France
Email: yves.sorel@inria.fr

Abstract

Partial answers have been provided in the real-time literature to the question whether preemptive systems are better than non-preemptive systems. This question has been investigated by many authors according to several points of view and it still remains open. Compared to preemptive real-time scheduling, non-preemptive real-time scheduling and the corresponding schedulability analyses have received considerable less attention in the research community. However, non-preemptive scheduling is widely used in industry, and it may be preferable to preemptive scheduling for numerous reasons. This approach is specially well suited in the case of hard real-time systems on the one hand where missing deadlines leads to catastrophic situations, and on the other hand where resources must not be wasted. In this paper, we firstly present the non-preemptive model of task with strict period, then we propose a schedulability condition for a set of such tasks, and finally we give a scheduling heuristic based on this condition.

1 Introduction

The main goal of hard real-time systems is to guarantee the schedulability of the task set on an uniprocessor platform so that each task completes its execution within its deadline. After the pioneering work of Liu and Layland [1], a lot of works has been done in the area of hard real-time scheduling to analyze and predict the schedulability of a preemptive task set under different scheduling policies and several task models. Although preemptive sheduling is more efficient than non-preemptive scheduling, this latter is important for a variety of reasons. Non-preemptive scheduling algorithms are easier to implement than preemptive algorithms, and can exhibit lower overhead at run-time. Preemption destroys program locality and affects the cache behavior, making the execution times more difficult to characterize and predict [2, 3].

Although, some works allow the computation of the exact cost of preemptions in the scheduling analysis [4], usually this cost is approximated as stated by Liu and Layland [1]. This approximation may lead to incorrect behaviour during the real-time execution of the tasks or at least a waste of resources due to the WCET and memory margins the designer must take. In the same vein the overhead of preemptive scheduling algorithms is more difficult to characterize and predict than the one of non-preemptive scheduling algorithms. Since scheduling overhead is often ignored in scheduling models, an implementation of a non-preemptive scheduler will be closer to the formal model than an implementation of a preemptive scheduler. In this case, the cost of the scheduler itself could be taken into account in schedulability conditions. Non-preemptive scheduling on a uniprocessor naturally guarantees exclusive access to shared resources and data, thus eliminating both the need for synchronization and its associated overhead. In control applications, the input-output delay and jitter are minimized for all tasks when using a non-preemptive scheduling discipline, since the interval between the start and end times is always equal to the task computation time [5]. This simplifies the techniques for delay compensation in the control design. In many practical real-time scheduling problems involving I/O scheduling, properties of device hardware and software either make preemption impossible of prohibitively expensive [6]. For these reasons, designers often use non-preemptive approaches even if the theoretical results do not extend easily to them [7].

In hard real-time systems some sensors and actuators have accurate periods. In order to produce (resp. receive) data at the right period the corresponding real-time tasks must have strict periods. Strict period means that if the task τ_i has the period T_i then $\forall j \in \mathbb{N}, S_i^{j+1} - S_i^j = T_i$ [8], where τ_i^j and τ_i^{j+1} are respectively the i^{th} and the $(i+1)^{th}$ repetitions of the task τ_i that we call instances, and S_i^j and S_i^{j+1} are respectively their start times. On the other hand, these sensor and actuator tasks always cooperate with other tasks the periods of which may be strict or not. In this paper, in order to simplify the problem, we shall assume that all the periods are strict rather than a mixture of strict and non strict periods. In order to schedule a set of non-preemptive strict periodic tasks, it

is enough to study the behaviors of these tasks for a time interval equal to the least common multiple (LCM), called the hyper-period [9].

It exists a lot of uniprocessor schedulability analyses based on scheduling algorithms like RM and EDF, but as they deal with non strict periods, their schedulability conditions become at least a necessary conditions in the case of strict periods.

This paper is organized as follows: in section 2 we present the related work. Section 3 is devoted to the schedulability analysis: we start by presenting the model of tasks then we propose a schedulability analysis through several theorems and corollaries. Section 4 gives a scheduling heuristic, and finally, section 5 presents a conclusion and further work.

2 Related work

Preemption related problems have received considerable attention in the real-time community. For example it exists a lot of uniprocessor schedulability conditions for popular algorithms like RM and EDF [1]. Unfortunately, these schedulability conditions become, at best, necessary conditions [10] in the non-preemptive case. However, non-preemption related problems must not be ignored since their resolutions may have great advantages in term of schedulability as pointed out previously. On the other hand these problems are NP-Hard in the strong sense as Jeffay, Stanat and Martel [6] showed. Baruah and Chakraborty [11] analyzed the schedulability of the non-preemptive recurring task model and showed that there exists polynomial time approximation algorithms for both preemptive and non-preemptive scheduling. Buttazzo and Cervin [5] used the non-preemptive task model to reduce jitter. A comprehensive schedulability analysis of non-preemptive systems was performed by George, Rivierre, and Spuri [10]. The main difference between the works previously presented and the works proposed in this paper lies in the type of period we consider. We remind the reader that usually periods are such that the difference between the start times of two task instances may vary whereas it is a constant in our case.

Cucu and al. [12, 13, 14] extended the result given by George and al. [10] in two directions: first when some non-preemptive tasks with strict periods have precedences, and second when multiple pairs of such tasks have latency constraints. These multiple latency constraints also called "end-to-end" constraints, possibly may have themselves precedences. In the same context (non-preemptive tasks with strict periods and precedences), Kermia and al. gave in [15] a necessary and sufficient schedulability condition for two tasks, which becomes a sufficient condition for more than two tasks.

Eisenbrand and al. proposed a similar works in [16]. They studied the schedulability conditions of tasks which have harmonic periods, i.e., for each pair (τ_i, τ_j), $T_i \mid T_j$ or $T_j \mid T_i$. This problem is a particular case of the general problem presented in this paper.

3 Schedulability analysis

As the problem of scheduling a set of non-preemptive periodic tasks on an uniprocessor is NP-Hard in the strong sense, we propose a scheduling heuristic based on a local schedulability condition. This local condition assumes that a set of tasks is already scheduled and verifies if a new task added to this set leads to a new schedulable set of tasks.

We start by studying the problem of two tasks, one already scheduled and a new task to be scheduled, then we extend this result for a set of more than two tasks.

3.1 Tasks model

We consider real-time systems of non-preemtive tasks with strict periods. We assume that every task has a deadline equal to its period. A non-preemptive task $\tau_i = (C_i, T_i, S_i)$ with the strict period T_i is characterized by:

- a period T_i equal to the deadline,
- a worst case execution time $C_i \leq T_i$,
- a start time S_i.

Afterwards, when the start time is unknown a task $\tau_i = (C_i, T_i, S_i)$ is denoted by $\tau_i = (C_i, T_i)$. We denote by S_i the start time of the first instance of a task τ_i: $S_i = S_i^1$.

The figure 1 shows an example of task with strict period.

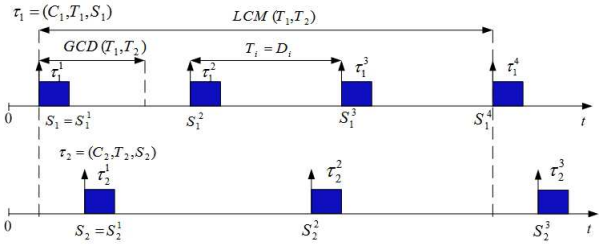

Figure 1: Model for non-preemptive tasks with strict period

We assume that periods and WCETs are multiple of a unit of time U, i.e. they are integers representing some cycles of the processor clock. If a task τ_i with execution time C_i is said to start at time unit t, it starts at the beginning of time unit t and completes at the end of time unit $t + C_i - 1$. Reciprocally, a time interval $[t_1, t_2]$ denotes a set of consecutive time units, given by $\{t_1, t_1 + 1, ..., t_2\}$.

3.2 Schedulability analysis of two tasks

Before giving schedulability and non-schedulability conditions for two tasks, we start by introducing some intermediate results.

The next theorem presents a necessary and sufficient condition for the schedulability of two tasks $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2)$ when the great common divisor (GCD) of the two task periods $GCD(T_1, T_2)$ is added to the start time $S_i,\ i = 1, 2$ of a task $\tau_i,\ i = 1, 2$.

Theorem 1 *Two tasks $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2)$ are schedulable if and only if the tasks $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2' = (C_2, T_2, S_2')$ are schedulable, where $S_2' = S_2 + g$ and $g = GCD(T_1, T_2)$.*

Proof
We start by proving the sufficient condition. We assume that the tasks $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2)$ are schedulable. Let ΔS_1 (resp. ΔS_2) be the time interval between the start times of τ_2 and τ_1 (resp. τ_2' and τ_1): $\Delta S_1 = S_2 - S_1$ and $\Delta S_2 = (S_2 + g) - S_1$. Lets ΔS be the time interval between an instance of τ_2' and an instance of τ_1: $\Delta S = (S_2' + n \cdot T_2) - (S_1 + m \cdot T_1)$, where $n, m \in \mathbb{N}$.

$$\begin{aligned}\Delta S &= (S_2 + g + n \cdot T_2) - (S_1 + m \cdot T_1) \\ &= (S_2 - S_1) + n \cdot T_2 - m \cdot T_1 + g \\ &= \Delta S_1 + n \cdot T_2 - m \cdot T_1 + g.\end{aligned}$$

According to Bezout's theorem [17], $\exists p, q \in \mathbb{Z}$ such that $p \cdot T_2 + q \cdot T_1 = g$, where $g = GCD(T_1, T_2)$. By choosing $p = -n$ and $q = m$ we have: $\Delta S = \Delta S_1 - (p \cdot T_2 - q \cdot T_1 + g) = \Delta S_1$.

So $\exists n, m \in \mathbb{N} :\ \Delta S = \Delta S_1$, which means that the the difference between the start time of the first instances of τ_1 and τ_2 is equal to the difference between the n^{th} instance of the task τ_1 and the m^{th} instance of the task τ_2'. As τ_1 and τ_2 are schedulable then τ_1 and τ_2' are also schedulable.

To prove the necessary condition, we assume that $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2' = (C_2, T_2, S_2')$ are schedulable, with $S_2' = S_2 + GCD(T_1, T_2)$.

Let $\Delta S = (S_2 + n \cdot T_2) - (S_1 + m \cdot T_1)$.

$$\begin{aligned}\Delta S &= (S_2 + (g - g) + n \cdot T_2) - (S_1 + m \cdot T_1) \\ &= (S_2 + g - S_1) + n \cdot T_2 - m \cdot T_1 - g \\ &= \Delta S_2 + n \cdot T_2 - m \cdot T_1 - g.\end{aligned}$$

According to the Bezout theorem $\exists n, m \in \mathbb{N}/n \cdot T_2 - m \cdot T_1 - g = 0$, so $\exists n, m \in \mathbb{N},\ \Delta S = \Delta S_1$ then the tasks τ_1 and τ_2 are schedulable ⊡

The next corollary is a direct deduction of the theorem 1. It shows that two tasks $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2)$ remain schedulable if

$$\forall n \in \mathbb{N},\ n \cdot GCD(T_1, T_2)$$

is added to the start time S_i of a task τ_i.

Corollary 1 *Two tasks $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2)$ are schedulable if and only if $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2' = (C_2, T_2, S_2')$ are schedulable, where $S_2' = S_2 + n \cdot g$, $n \in \mathbb{N}$ and $g = GCD(T_1, T_2)$.*

Proof
We prove this corollary by recurrence.
For $n = 1$, this corollary is equivalent to the theorem 1. For $n \geq 2$ we assume that $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2' = (C_2, T_2, S_2 + n.g)$ are schedulable. According to the theorem 1, $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2' = (C_2, T_2, (S_2 + n \cdot g) + g) = (C_2, T_2, S_2 + (n+1) \cdot g))$ are also schedulable, so $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2' = (C_2, T_2, S_2 + (n + 1) \cdot g)$ are schedulable ⊡

Remark 1 *The corollary 1 can be reformulated, by replacing n from corrolary 1 by 1, as follows:*
Two tasks $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2)$ are schedulable if and only if $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2 mod(GCD(T_1, T_2)))$ are schedulable.
Where $mod(GCD(T_1, T_2))$ is the modulo function of $GCD(T_1, T_2)$.

Using the previous results, the following theorem gives a necessary and sufficient condition of the schedulability for two tasks $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2)$.

Theorem 2 *Two tasks $\tau_1 = (C_1, T_1, S_1)$ and $\tau_2 = (C_2, T_2, S_2)$ are schedulable if and only if*

$$C_1 \leq (S_2 - S_1)\ mod(g)) \leq g - C_2 \tag{1}$$

where $g = GCD(T_1, T_2)$.

Proof
We start by proving that the condition (1) is a sufficient condition. Let $T_1 = n_1 \cdot g$ and $T_2 = n_2 \cdot g$ where $g = GCD(T_1, T_2)$ and $n_1 \wedge n_2 = 1$. Without loss of generality, we assume that $S_1 = 0$.

The condition (1) becomes

$$C_1 \leq S_2\ mod(g) \leq g - C_2. \tag{2}$$

According to the remark 1, the condition (2) becomes

$$C_1 \leq S_2 \leq g - C_2. \tag{3}$$

Each instance of the task τ_1 is executed in the interval

$$I_1 = \bigcup_{n=0}^{+\infty} [n \cdot T_1,\ n \cdot T_1 + C_1]$$

$$= \bigcup_{n=0}^{+\infty} [(n \cdot n_1) \cdot g,\ (n \cdot n_1) \cdot g + C_1]$$

and each instance of the task τ_2 is executed in the interval

$$I_2 = \bigcup_{m=0}^{+\infty} [m \cdot T_2 + S_2,\ m \cdot T_2 + S_2 + C_2]$$

$$= \bigcup_{m=0}^{+\infty} [(m \cdot n_2) \cdot g + S_2,\ (m \cdot n_2) \cdot g + S_2 + C_2].$$

For each interval of time of length g, an instance of τ_1 is executed in the interval $[0, C_1]$ and an instance of τ_2 is executed in the interval $[S_2, S_2 + C_2]$.

The condition (3) gives: $C_1 \leq S_2$ and $S_2 + C_2 \leq g$. So $[0, C_1] \cap [S_2, S_2 + C_2] = \emptyset$ then the tasks τ_1 and τ_2 are schedulable.

To prove the necessity of the condition (1), we show that if the condition (1) is not satisfied then the tasks τ_1 and τ_2 are not schedulable. Without loss of generality, we assume that $S_1 = 0$. The condition (1) is not satisfied means that:

$$S_2\ mod(g) < C_1 \quad or \quad g - C_2 < S_2\ mod(g).$$

According to the remark 1, the last condition becomes:

$$S_2 < C_1 \quad or \quad g - C_2 < S_2.$$

The first condition $S_2 < C_1$ means that an instance of the task τ_2 starts its execution before the end of the execution of the instance of the task τ_1, then the two tasks τ_1 and τ_2 are not schedulable.

The second condition $g - C_2 < S_2$ becomes $g < S_2 + C_2$, which means that an instance of the task τ_2 completes its execution outside a time interval of length g, so it will overlap an instance of τ_1 which start it execution exactly at the bigenning of a time interval of length equal to g, then the τ_1 and τ_2 are not schedulable.

So in these two last cases the tasks τ_1 and τ_2 are not schedulable, which prove the necessety of the condition (1) ⊡

Example 1 *Let consider two tasks* $\tau_1 = (1, 8, 0)$ *and* $\tau_1 = (2, 12, 5)$. $GCD(T_1, T_2) = GCD(8, 12) = 4$, $GCD(T_1, T_2) - C_2 = 2$ *and* $(S_2 - S_1)\ mod(GCD(T_1, T_2)) = 5\ mod(4) = 1$. *So the condition (1) is satisfied then the tasks* τ_1 *and* τ_2 *are schedulable (figure 2).*

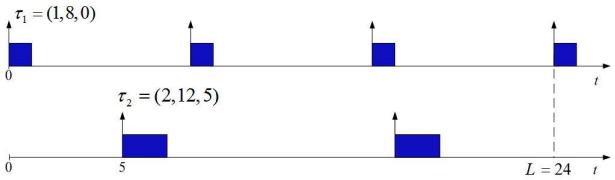

Figure 2: Scheduling of two tasks

The following corollary gives a specific condition of schedulability for two tasks. Notice that in this corollary the start times are useless because two tasks can be schedulable with many different start times.

Corollary 2 *Two tasks* $\tau_1 = (C_1, T_1)$ *and* $\tau_2 = (C_2, T_2)$ *are schedulable if and only if:*

$$C_1 + C_2 \leq GCD(T_1, T_2) \tag{4}$$

Proof
From the condition (1) of the theorem 2 we have:

$$C_1 \leq GCD(T_1, T_2) - C_2$$

then

$$C_1 + C_2 \leq GCD(T_1, T_2)$$

⊡

This corollary was presented in [15] as a general necessary and sufficient schedulability condition. We prove here that it is only a specific case of theorem 2.

The theorem 2 gives a time intervals for the schedulability of two tasks. The next theorem gives a non schedulability condition based on the computation of complementary intervals to the intervals where two tasks are schedulables.

Theorem 3 *Two tasks* $\tau_1 = (C_1, T_1, S_1)$ *and* $\tau_2 = (C_2, T_2, S_2)$ *are not schedulable if and only if:*

$$(S_2 - S_1) mod\,(GCD(T_1, T_2)) \in$$

$$[0\ ,\ C_1[\cup](GCD(T_1, T_2) - C_2)\ ,\ GCD(T_1, T_2)[\tag{5}$$

Proof
Two tasks are not schedulable if and only if the schedulability condition of the theorem 2 is not satisfied, which means that:

$$C_1 \geq (S_2 - S_1)\ mod(GCD(T_1, T_2)) \tag{6}$$

or

$$(S_2 - S_1)\ mod(GCD(T_1, T_2)) \geq GCD(T_1, T_2) - C_2. \tag{7}$$

As

$$0 \leq (S_2 - S_1)\ mod(GCD(T_1, T_2)) \leq GCD(T_1, T_2)$$

then the condition (6) becomes

$$0 \leq (S_2 - S_1)\ mod(GCD(T_1, T_2)) \leq C_1$$

and the condition (7) becomes

$$GCD(T_1, T_2) - C_2 \leq (S_2 - S_1)\ mod(GCD(T_1, T_2))$$

$$(S_2 - S_1)\ mod(GCD(T_1, T_2)) \leq GCD(T_1, T_2).$$

And finaly

$$(S_2 - S_1) mod\,(GCD(T_1, T_2)) \in$$

$$[0\ ,\ C_1[\cup](GCD(T_1, T_2) - C_2)\ ,\ GCD(T_1, T_2)[$$

⊡

Example 2 *Let consider the same tasks of the example 1 and change the start time of the task τ_2: $\tau_1 = (1, 8, 0)$ and $\tau_2 = (2, 12, 3)$. We have:*

$$(S_2 - S_1) mod(GCD(T_1, T_2)) = 3 mod(4) = 3$$

and the time interval

$$](GCD(T_1, T_2) - C_2), GCD(T_1, T_2)[=](4 - 2), 4[$$
$$=]2, 4[.$$

The condition (5) is satisfied then the tasks τ_1 and τ_2 are not schedulable. As we can see in the figure 3, the third instance τ_1^3 of the task τ_1 starts its execution before that the second instance τ_2^2 of the task τ_2 has completed its execution.

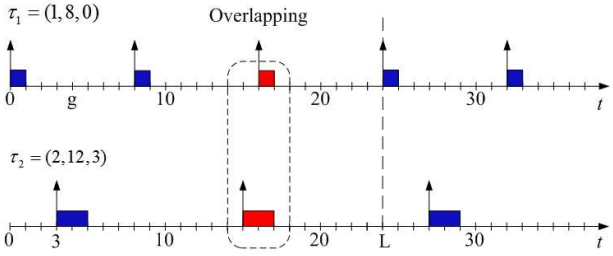

Figure 3: Overlapping of two tasks

The following corollary gives a specific condition of non schedulability for two tasks.

Corollary 3 *A set of n tasks $\Gamma_n = \{\tau_1 = (C_i, T_i), i = 1, n\}$ is not schedulable if at least two tasks τ_i and τ_j of Γ_n have comprime periods:*

$$\exists (i, j) \in [1, n]^2 \text{ such as } GCD(T_i, T_j) = 1 \quad (8)$$

Proof
If $\exists (i, j) \in [1, n]^2$, $GCD(T_i, T_j) = 1$:
The condition (5) of non schedulability of two tasks gives:

$$[0, C_1[\cup](GCD(T_1, T_2) - C_2), GCD(T_1, T_2)[=$$
$$[0, C_1[\cup]1 - C_2, 1[$$

and

$$(S_2 - S_1) mod (GCD(T_1, T_2)) = 0.$$

Thus the condition (5) is satisfied so τ_1 and τ_2 are not schedulable ⊡

This corollary was presented in [18] as a general necessary and sufficient non schedulability condition. We prove here that it is not true because it is only a specific case of theorem 3 which gives a general necessary and sufficient non schedulability condition.

The next theorem gives the possible start times of the second task to be scheduled when the first task is already scheduled.

Theorem 4 *Let $\tau_1 = (C_1, T_1)$ be a task already scheduled and $\tau_2 = (C_2, T_2)$ the task to be scheduled. If $\tau_1 = (C_1, T_1)$ and $\tau_2 = (C_2, T_2)$ are schedulable then the possible start times S_2 of τ_2 are given by: $\forall n \in \mathbb{N}$ and $\forall m \in [C_1, GCD(T_1, T_2) - C_2]$*

$$S_2 = S_1 + n \cdot GCD(T_1, T_2) + m \quad (9)$$

Proof
Let assume that a task $\tau_1 = (C_1, T_1)$ is already scheduled and a task $\tau_2 = (C_2, T_2)$ is a task to be scheduled. According to the theorem 2, $\tau_1 = (C_1, T_1)$ and $\tau_2 = (C_2, T_2)$ are schedulable if and only if:

$$C_1 \leq (S_2 - S_1) \ mod(g)) \leq g - C_2$$

where $g = GCD(T_1, T_2)$. Let $(S_2 - S_1) \ mod(g) = n$ with $C_1 \leq n \leq g - C_2$
which is equivalent to:
$S_2 - S_1 = n + m \cdot g$ with $C_1 \leq n \leq g - C_2$ and $m \in \mathbb{N}$
so $S_2 = S_1 + n + m \cdot g$ with $C_1 \leq n \leq g - C_2$ and $m \in \mathbb{N}$ ⊡

Example 3 *Let consider a task $\tau_1 = (1, 10)$ already scheduled and $\tau_2 = (3, 15)$ the task to be scheduled. We assume that the start time of τ_1 is $S_1 = 0$. As $GCD(T_1, T_2) = 5$ and $C_1 + C_2 = 4 \leq 5$ so the condition (4) is satisfied then the tasks τ_1 and τ_2 are schedulable. The theorem 5 gives:*
$S_2 = 0 + n \cdot 5 + m$ where $n \in \mathbb{N}$ and $1 \leq m \leq 5 - 3$.
So $S_2 = 5 \cdot n + m$ where $n \in \mathbb{N}$ and $1 \leq m \leq 2$.
S_2 belongs to the following set:

$$S_2 \in \{1, 2, 6, 7, 11, 12, 16, 17, \cdots\}.$$

The figure 4 shows some possible start times of the task τ_2.

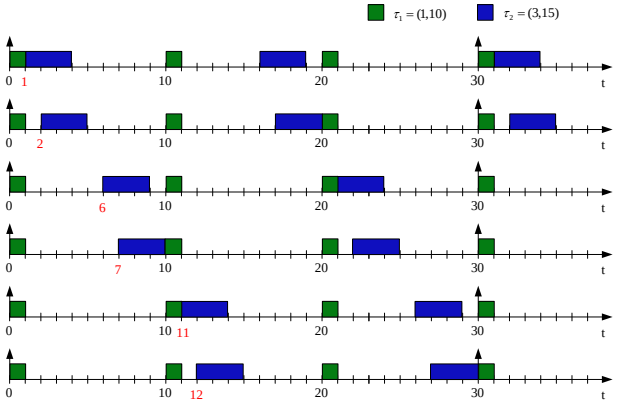

Figure 4: Scheduling possibilities for two tasks

3.3 Schedulability analysis for more than two tasks

Contrary to the schedulability study for two tasks where a necessary and sufficient condition of schedulability ex-

ists, there is no necessary and sufficient condition for more than two tasks [15].

It has been proven in [15] that the necessary and sufficient condition of schedulability for two tasks (4) given in the corollary 2 becomes a sufficient condition in the case of more than two tasks:

$$\sum_{i=1}^{n} C(t_i) \leq GCD(\forall i, T_i) \tag{10}$$

However, it is a very restrictive condition of schedulability for a set of tasks. The next example illustrates this restriction.

Example 4 *Let consider four tasks* $\tau_1 = (1, 6)$, $\tau_2 = (1, 8)$, $\tau_3 = (1, 12)$ *and* $\tau_4 = (1, 24)$. *We have* $g = GCD(T_1, T_2, T_3, T_4) = 2$. *These four tasks do not satisfy the condition (10)*

$$\sum_{i=1}^{4} C_i = 4 > g.$$

However, this set of tasks is schedulable as shown in the figure 5.

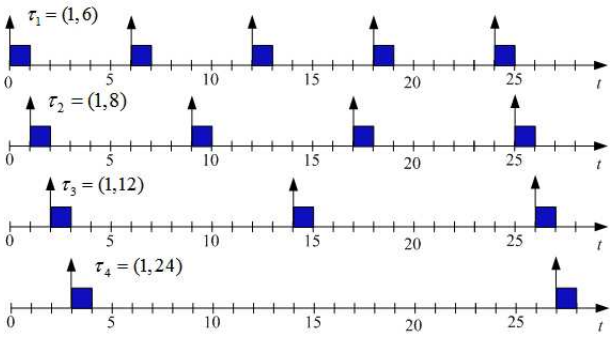

Figure 5: Scheduling of four tasks

The next theorem gives a schedulability condition for a set of tasks that does not satisfy the condition (10).

Theorem 5 *Let* $\Gamma_n = \{\tau_i = (C_i, T_i), i = 1, n\}$ *be a set of tasks that satisfies the condition (10). Let* τ_c *be the task to be scheduled such as* $\Gamma_n \cup \{\tau_c\}$ *does not satisfy the condition (10).* τ_c *is schedulable if:* $\exists \tau_i = (C_i, T_i) \in \Gamma_n$, $T_i \neq g$ *such as*

$$C_i \cdot \delta\left(T_c mod(T_i)\right) \geq C_c \tag{11}$$

Where mod *is the* $modulo$ *function and* δ *is the Kronecker symbol:*

$$\delta[i] = \begin{cases} 1 \ if \ \ i = 0 \\ 0 \ \ otherwise \end{cases}$$

Proof
The condition (11) is equivalent to

$$T_c \ is \ a \ multiple \ of \ T_i \quad and \quad C_i \geq C_c.$$

We assume that the tasks of Γ_n satisfy the condition (10) and all their first instances τ_i^1 are scheduled in the time interval $[0, g]$. In a time interval $[n \cdot g, (n+1) \cdot g]$, an instance τ_i^j is executed during the interval $[n \cdot g + S_i, (n+1) \cdot g + S_i + C_i]$, and no other instance of Γ_n could be executed in this interval. As we supposed that $T_i > g$ then some intervals of length g do not contain any instance of τ_i. In these intervals we can schedule a task which has the same period and WCET as τ_c. As such a task can be scheduled thus the task τ_c which has the same or a multiple period as τ_i and a worst execution time $C_c \leq C_i$, can be scheduled ⊡

Theorem 6 *Let* $\Gamma_n = \{\tau_i = (C_i, T_i), i = 1, n\}$ *be a set of tasks that satisfies the condition (10). Let* τ_c *be the task to be scheduled such as* $\Gamma_n \cup \{\tau_c\}$ *do not satisfy the condition (10).* τ_c *is schedulable if:* $\exists \tau_i = (C_i, T_i) \in \Gamma_n$, $T_i > g$ *such as*

$$C_i \cdot \delta\left(T_c \cdot mod(2g) + T_i \cdot mod(2g)\right) \geq C_c \tag{12}$$

Proof
The condition (12) is equivalent to:

$$T_i \ and \ T_c \ are \ multiple \ of \ 2 \cdot g \quad and \quad C_i \geq C_c.$$

We assume that the tasks of Γ_n satisfy the condition (10) and all their first intances τ_i^1 are scheduled in the time interval $[0, g]$.

Let $T_i = 2 \cdot n_i \cdot g$ and $T_c = 2 \cdot n_c \cdot g$.

Let $S_c = S_i + g$.

The start times of the instances τ_i^j and τ_c^j are given by

$$S_i^j = S_i + n \cdot T_i = S_i + 2n \cdot n_i \cdot g, \ n \in \mathbb{N}$$

and

$$S_c^j = S_i + g + m \cdot T_c = S_i + g + 2m \cdot n_c \cdot g, \ n \in \mathbb{N}.$$

The instances τ_c^j and τ_i^j overlap if $S_c^j = S_i^j$ then $\exists n, m \in \mathbb{N}$, $S_i + g + 2m \cdot n_c \cdot g = S_i + 2n \cdot n_i \cdot g$ which gives $2m \cdot n_c + 1 = 2n \cdot n_i$ or $2(n \cdot n_i - m \cdot n_c) = 1$ that is impossible, so no instance τ_c^j overlaps τ_i^j, then the task τ_c is schedulable ⊡

Corollary 4 *Let* $\Gamma_n = \{\tau_i = (C_i, T_i), i = 1, n\}$ *be a set of tasks that satisfy the condition (10). Let* τ_c *be the task to be scheduled.* τ_c *is schedulable if:*

$$\left[g - \sum_{i=1}^{n} C_i\right] \cdot \delta\left[T_c \cdot mod(g)\right] + \sum_{i=1}^{n} C_i \cdot \delta\left[T_c mod(T_i) \cdot (T_c \cdot mod(2g) + T_i \cdot mod(2g))\right] \geq C_c. \tag{13}$$

Proof
The condition (13) can be written as

$$[g - \sum_{i=1}^{n} C_i] \cdot \delta\left[T_c \cdot mod(g)\right] + \sum_{i=1}^{n} C_i \cdot \delta\left[T_c mod(T_i)\right] \cdot \delta\left[T_c \cdot mod(2g) + T_i \cdot mod(2g)\right] \geq C_c. \quad (14)$$

Let $\tau_c = (C_c, T_c)$ the task to be scheduled.

If $g - \sum_{i=1}^{n} C_i > 0$ and T_c is a multiple of g then the task $\tau'_c = ((g - \sum_{i=1}^{n} C_i), Tc)$ is schedulable.

If $\exists \tau_i \in \Gamma_n$ such as $T_c mod(T_i) = 0$ or $T_c mod(2g) + T_i \cdot mod(2g) = 0$ then the task $\tau''_c = (C_i, T_c)$ is schedulable.

Let Γ_s be the set of tasks $\tau = (C_i, T_c)$ which are schedulable. If $\sum_{\tau_{c_i} \in \Gamma_s} C_i \geq C_c$ than the task $\tau_c = (C_c, T_c)$ is schedulable ⊡

The following section presents the scheduling heuristic based on this schedulability condition.

4 Scheduling heuristic

Now, we propose a scheduling heuristic based on the schedulability condition presented in the previous section. First, the heuristic (algorithm 1) initializes the set Γ with the system of n tasks to schedule, and the sets Γ', Γ'', Γ''' and Γ_s with the empty set. Γ' will contain the schedulable tasks that satisfy the condition (10) and $\Gamma'' = \Gamma \setminus \Gamma'$ will contain the relative complement set of Γ' in Γ. Γ_s will contain the schedulable tasks that satisfy the condition (13). Γ''' is used for temporary computations. The heuristic builds iteratively, according to the index k of the task τ_k from Γ, the set Γ' of tasks which satisfies the condition (10). Then it creates the set $\Gamma'' = \Gamma \setminus \Gamma'$, and it applies iteratively, according to the index i of the task τ_i from Γ'', the schedulability condition (13) as follows. To schedule a task τ_i from Γ'', the heuristic computes iteratively, according to the index j of the task τ_j from Γ', $C_j \cdot \delta\left[T_i mod(T_j)\right] \cdot \delta\left[T_i \cdot mod(2g) + T_j \cdot mod(2g)\right]$, and adds them to $g - \sum_{i=1}^{n} C_i$. If the condition (13) is satisfied, that is the result of the previous summation is greater or equal to the C_i of the task τ_i, then this latter task and the tasks τ_j which satisfy the condition $T_i \cdot mod(g) \cdot (T_i \cdot mod(2g) + T_j \cdot mod(2g)) = 0$, are moved to the set Γ_s. The heuristic stops when Γ' or Γ'' becomes empty. Finally, if the set Γ'' is empty then the initial set Γ, else it is not schedulable but the set $\Gamma_s \cup \Gamma'$ is schedulable.

The following example illustrates the execution of our heuristic in the case of a set of four tasks.

Example 5 *Let consider a set of six tasks* $\Gamma = \{\tau_1 = (1, 12),\ \tau_2 = (3, 16),\ \tau_3 = (1, 20),\ \tau_4 = (2, 24),\ \tau_5 = (1, 40)\}$, $g = GCD(12, 16) = 4$ *and* $C_1 + C_2 = 4$. *The condition (4) gives:* $C_1 + C_2 = GCD(T_1, T_2)$ *so it does not give any information about the schedulability of the rest of the tasks of* Γ.

The set of tasks Γ' *is initialized by* $\Gamma' = \{\tau_1 = (1, 12),\ \tau_2 = (3, 16)\}$.

$\Gamma'' = \{\ \tau_3 = (1, 20),\ \tau_4 = (2, 24),\ \tau_5 = (1, 40)\}$.

Let apply the condition (13) to the rest of tasks of Γ''.

The first term of the condition (13) becomes:

$$\left[g - \sum_{i=1}^{n} C_i\right] \cdot \delta\left[T_c \cdot mod(g)\right] = (4 - 1 - 3) \cdot \delta\left[T_c \cdot mod(g)\right] = 0.$$

For $\tau_c = \tau_3 = (1, 20)$, *the condition (13) gives:*

$$\sum_{i=1}^{2} C_i \cdot \delta\left[T_c mod(T_i) \cdot (T_c \cdot mod(2g) + T_i \cdot mod(2g))\right]$$

$$= C_1 \cdot \delta\left[T_c mod(T_1) \cdot (T_c \cdot mod(2g) + T_1 \cdot mod(2g))\right]$$

$$+ C_2 \cdot \delta\left[T_c mod(T_2) \cdot (T_c \cdot mod(2g) + T_2 \cdot mod(2g))\right]$$

$$= 1 \cdot \delta\left[20 mod(12) \cdot (20 \cdot mod(2 \cdot 4) + 12 \cdot mod(2 \cdot 4))\right]$$

$$+ 3 \cdot \delta\left[20 mod(16) \cdot (20 \cdot mod(2 \cdot 4) + 16 \cdot mod(2 \cdot 4))\right]$$

$$= 1 \cdot \delta[8 \cdot (4 + 4)] + 3 \cdot \delta[4 \cdot (4 + 0)] = 0 \ngeq C_3 = 1.$$

The condition (13) is not satisfied so the task τ_3 *is not schedulable.* τ_3 *is removed from* Γ''*:* $\Gamma'' = \{\ \tau_4 = (2, 24),\ \tau_5 = (1, 40)\}$.

For $\tau_c = \tau_4 = (2, 24)$, *the condition (13) gives:*

$$1 \cdot \delta\left[24 mod(12) \cdot (24 \cdot mod(2 \cdot 4) + 12 \cdot mod(2 \cdot 4))\right]$$

$$+ 3 \cdot \delta\left[24 mod(16) \cdot (24 \cdot mod(2 \cdot 4) + 16 \cdot mod(2 \cdot 4))\right]$$

$$= 1 \cdot \delta\left[0\right] + 3 \cdot \delta\left[0\right] = 4 > C_4 = 2$$

The condition (13) is satisfied so the task τ_4 *is schedulable.* τ_4 *is removed from* Γ'' *to* Γ_s *and* τ_1 *is removed from* Γ' *to* Γ_s*:* $\Gamma_s = \{\tau_1 = (1, 12), \tau_4 = (2, 24)\}$ *and* $\Gamma'' = \{\tau_5 = (1, 40)\}$. *and* $\Gamma' = \{\tau_2 = (1, 40)\}$

For $\tau_c = \tau_5 = (1, 40)$, *the condition (13) gives:*

$$3 \cdot \delta\left[40 mod(16) \cdot (40 \cdot mod(2 \cdot 4) + 16 \cdot mod(2 \cdot 4))\right]$$

$$3 \cdot \delta\left[0\right] = 3 > C_5 = 1$$

The condition (13) is satisfied so the task τ_5 *is schedulable, then:* $\Gamma_s = \{\tau_1 = (1, 12), \tau_2 = (1, 40), \tau_4 = (2, 24)\}, \tau_5 = (1, 40)$ *and* $\Gamma'' = \emptyset$ *and* $\Gamma' = \emptyset$. *The initial set* Γ *is not schedulalbe but the set* $\Gamma'' \cup \Gamma_s = \Gamma_s$ *is schedulable.*

Algorithm 1 Scheduling Algorithm

Initialization of the set Γ with the n tasks to be scheduled. Initialization with the empty set of the set Γ' which will contain the tasks satisfying the condition (10), of the set Γ'' which will contain the tasks which does not satisfy the condition (10), of the set Γ_s which will contain the schedulable tasks that satisfies the condition (13), and of the set Γ''' which will contain temporary tasks.
$\Gamma' = \{\tau_1\}$ where $\tau_1 \in \Gamma$.
for $k = 2$ to n **do**
  $g = GCD(T_i, \tau_i \in \Gamma' \cup \{\tau_k\})$
  $C = C_k$.
  for $l = 1$ to $|\Gamma'|$ **do**
    $C = C + C_l$
  end for
  if condition (10) is satisfied, i.e. $C \leq g$ **then**
    Copy the task τ_k from Γ to Γ'.
  else
    Copy the task τ_k from Γ to Γ''.
  end if
end for
Let $m = |\Gamma'|$.
Let $g = GCD(\tau_i,\ \forall \tau_i \in \Gamma')$
Let $C' = g - \sum_{1}^{m} C_i$
Let $i = 1$
while $\Gamma' \neq \emptyset$ and $i \leq n - m$ **do**
  Let C=0
  Let i=1
  for $j = 1$ to m **do**
    if $T_i mod(T_j) \cdot (T_i mod(2g) + T_j mod(2g)) = 0$ **then**
      $C = C + C_j$
      Move the task τ_j from Γ' to Γ'''
    end if
  end for
  if $T_i \cdot mod(g) = 0$ **then**
    $C'' = C'$
  else
    $C'' = 0$
  end if
  if condition (13) is satisfied, i.e. $C'' + C \geq C_i$ **then**
    Move the task τ_i from Γ'' to Γ_s
    Move the tasks of Γ''' to Γ_s
  end if
  increment i
end while
if $\Gamma' \cup \Gamma_s = \Gamma$ **then**
  Γ is schedulable.
else
  Γ is not schedulable.
  $\Gamma' \cup \Gamma_s$ is schedulable.
end if

The figure 6 shows the result obtained with our scheduling heuristic for the set of tasks Γ. This result is displayed using the software SAS [19].

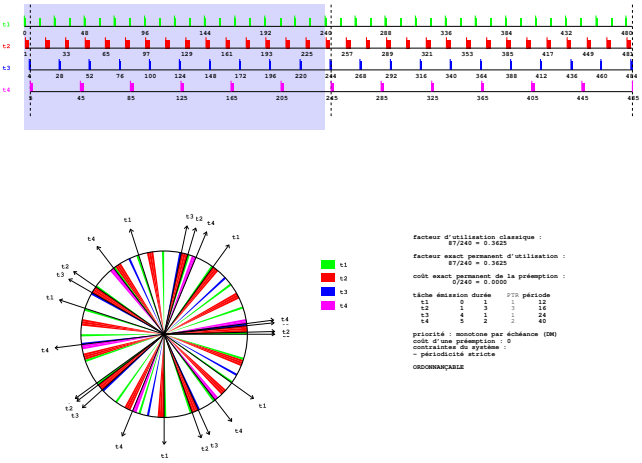

Figure 6: Scheduling of four tasks using the heuristic

5 Conclusion and further work

In this paper we present a schedulability analysis in the case of non-preemptive tasks with strict periods. We start by giving a schedulability condition for two tasks then we give the schedulability condition for a set of more than two tasks. We finally propose a scheduling heuristic which is based on this schedulability condition, where a set of tasks is already scheduled and a new task is to be scheduled.

Further work will propose a more general schedulability analysis in the case where the deadline is different from the period, since in this paper we addressed a schedulability analysis in the case where the period is equal to the deadline. Also, we plan to study the schedulability of non-preemptive tasks with strict and non-strict period.

References

[1] C. L.Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.

[2] H. Ramaprasad and F. Mueller. Tightening the bounds on feasible preemption points. In *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 212–224, Washington, DC, USA, 2006. IEEE Computer Society.

[3] H. Ramaprasad and F. Mueller. Bounding worst-case response time for tasks with non-preemptive regions. In *RTAS '08: Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 58–67, Washington, DC, USA, 2008. IEEE Computer Society.

[4] P. Meumeu Yomsi and Y. Sorel. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In *Proceedings of 19th Euromicro Conference on Real-Time Systems, ECRTS'07*, Pisa, Italy, July 2007.

[5] G. Buttazzo and A. Cervin. Comparative assessment and evaluation of jitter control methods. In *Proc. 15th International Conference on Real-Time and Network Systems*, Nancy, France, March 2007.

[6] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of period and sporadic tasks. In *Proceedings of the 12 th IEEE Symposium on Real-Time Systems*, pages 129–139, December 1991.

[7] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-vincentelli. Scheduling for embedded real-time systems. *IEEE Design and Test of Computers*, 15(1):71–82, 1998.

[8] L. Cucu, R. Kocik, and Y. Sorel. Real-time scheduling for systems with precedence, periodicity and latency constraints. In *Proceedings of 10th Real-Time Systems Conference, RTS'02*, Paris, France, March 2002.

[9] K. Danne and M. Platzner. A heuristic approach to schedule periodic real-time tasks on reconfigurable hardware. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 568–573, August 2005.

[10] L. George, N. Rivierre, and M. Spuri. Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling. Research Report RR-2966, INRIA, 1996. Projet REFLECS.

[11] S.K. Baruah and S. Chakraborty. Schedulability analysis of non-preemptive recurring real-time tasks. *Parallel and Distributed Processing Symposium, International*, 0:149, 2006.

[12] L. Cucu and Y. Sorel. Schedulability condition for systems with precedence and periodicity constraints without preemption. In *Proceedings of 11th Real-Time Systems Conference, RTS'03*, Paris, March 2003.

[13] L. Cucu, N. Pernet, and Y. Sorel. Periodic real-time scheduling: from deadline-based model to latency-based model. *Annals of Operations Research*, 2007.

[14] L. Cucu-Grosjean and Y. Sorel. A schedulability test for real-time dependant periodic task systems with latency constraints. In *Proceedings of conference Models and Algorithms for Planning and Scheduling Problems, MAPSP'09*, Abbey Rolduc, The Netherlands, July 2009.

[15] O. Kermia and Y. Sorel. Schedulability analysis for non-preemptive tasks under strict periodicity constraints. In *Proceedings of 14th International Conference on Real-Time Computing Systems and Applications, RTCSA'08*, Kaohsiung, Taiwan, August 2008.

[16] Friedrich Eisenbrand, Nicolai Hhnle, Martin Niemeier, Martin Skutella, Jos Verschae, and Andreas Wiese. Scheduling periodic tasks in a hard real-time environment. In *37th International Colloquium on Automata, Languages and Programming (ICALP2010)*, volume 37, pages 299–311. Springer-Verlag, 2010.

[17] D. Kirby. On bezout's theorem. *The Quarterly Journal of Mathematics*, Dec 1988.

[18] P. Meumeu Yomsi and Y. Sorel. Non-schedulability conditions for off-line scheduling of real-time systems subject to precedence and strict periodicity constraints. In *Proceedings of 11th IEEE International Conference on Emerging technologies and Factory Automation, ETFA'06, WIP*, Prague, Czech Republic, September 2006.

[19] P. Meumeu Yomsi, L. George, Y. Sorel, and D. de Rauglaudre. Improving the quality of control of periodic tasks scheduled by fp with an asynchronous approach. *International Journal on Advances in Systems and Measurements*, 2(2), 2009.